(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2000－20291
（P2000－20291A）
(43)公開日　平成12年１月21日(2000.1.21)

(51)Int.Cl.⁷	識別記号	ＦＩ		テーマコード（参考）
Ｇ０６Ｆ　9/06	５３０	Ｃ０６Ｆ　9/06	５３０Ｃ	５Ｂ０４２
			５３０Ｃ	５Ｂ０７６
11/28	３４０	11/28	３４０Ｃ	

審査請求　未請求　請求項の数19　ＯＬ　（全 19 頁）

(21)出願番号　特願平10－190815

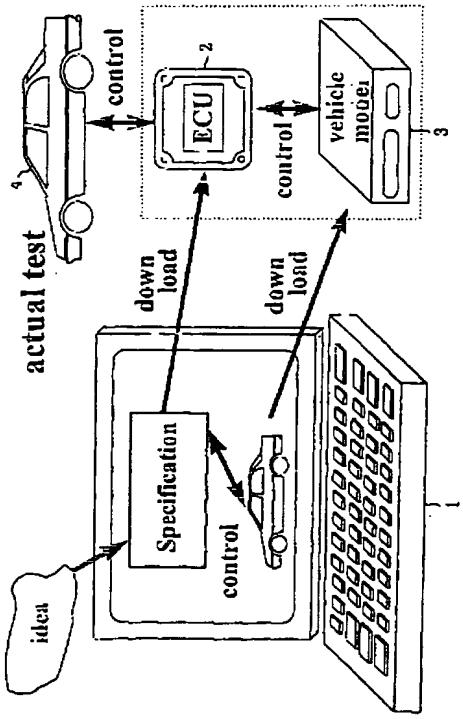
(22)出願日　平成10年７月６日(1998.7.6)

(71)出願人　000003207
トヨタ自動車株式会社
愛知県豊田市トヨタ町１番地
(72)発明者　足立　憲保
愛知県豊田市トヨタ町１番地　トヨタ自動車株式会社内
(74)代理人　100075258
弁理士　吉田　研二　（外２名）

Ｆターム(参考)　5B042 BB07
5B076 DD04 DE01 DE03 EC06

(54)【発明の名称】　車両用プログラム開発支援方法および装置

(57)【要約】
【課題】　車両制御プログラムの開発における工数を削減する。
【解決手段】　プログラム生成装置１は、データフローチャートおよびステートフローチャートのかたちで入力された制御仕様から車両用Ｃコードを自動生成する。車両用Ｃコードは車両ＥＣＵ２にダウンロードされる。車両モデル装置３を制御対象とする車両ＥＣＵ２の制御動作がプログラム生成装置１にモニタされ、デバッグ作業が行われる。制御仕様から自動生成した車両用Ｃコードを使うので確実なデバッグ作業が容易に行われる。また手作業のコーディング作業が不要になる。データフローチャートのシンボルブロックに整数情報をもたせることで、整数情報を利用して整数ロジックの車両用Ｃコードを自動生成できる。



【特許請求の範囲】
【請求項１】　入力された制御仕様から車両用コードを生成する機能をもつプログラム生成装置を用いて、車両制御プログラムを生成するプログラム生成ステップと、
生成された前記車両制御プログラムを車両ＥＣＵにダウンロードするダウンロードステップと、
前記車両ＥＣＵに前記車両制御プログラムを実行させて、前記車両制御プログラムのデバッグを行うデバッグステップと、
を含むことを特徴とする車両用プログラム開発支援方法。
【請求項２】　請求項１に記載の車両用プログラム開発支援方法において、
前記デバッグステップのデバッグ作業は、前記車両ＥＣＵが前記車両制御プログラムを実行した結果を監視する前記プログラム生成装置上で行われることを特徴とする車両用プログラム開発支援方法。
【請求項３】　請求項２に記載の車両用プログラム開発支援方法において、
制御対象の車両をモデル化した車両モデル装置に前記車両ＥＣＵを接続し、前記車両ＥＣＵに前記車両モデル装置を制御させる制御実行ステップと、
制御実行中の前記車両ＥＣＵおよび前記車両モデル装置を監視する監視ステップと、
を含むことを特徴とする車両用プログラム開発支援方法。
【請求項４】　請求項３に記載の車両用プログラム開発支援方法において、
入力された車両仕様に基づいて車両モデルを前記プログラム生成装置上で生成するモデル生成ステップと、
前記モデル生成ステップで生成された車両モデルを前記車両モデル装置にダウンロードするモデルダウンロードステップと、
を含むことを特徴とする車両用プログラム開発支援方法。
【請求項５】　請求項１～４のいずれかに記載の車両用プログラム開発支援方法において、
前記車両用コードは、車両ＥＣＵで処理される整数ロジックに適合するように一般コードを変形したものであることを特徴とする車両用プログラム開発支援方法。
【請求項６】　請求項５に記載の車両用プログラム開発支援方法において、
前記プログラム生成ステップでは、前記プログラム生成装置に入力された整数変換条件に従った車両制御プログラムが生成されることを特徴とする車両用プログラム開発支援方法。
【請求項７】　車両の制御仕様を入力する入力手段と、
前記制御仕様に基づいて、車両制御プログラムの車両用コードを生成するプログラム生成手段と、
前記車両制御プログラムを外部の車両ＥＣＵにダウンロードするダウンロード手段と、
前記車両ＥＣＵで前記車両制御プログラムを実行した実行結果を出力する出力手段と、
を含むことを特徴とする車両用プログラム開発支援装置。
【請求項８】　車両の制御仕様を表すデータフローチャートおよびステートフローチャートを作成するチャート作成機能と、
作成されたチャートに基づいて、車両ＥＣＵで処理される整数ロジックで構成された車両制御プログラムの車両用コードを生成するプログラムコード生成機能と、
を有することを特徴とする車両用プログラム開発支援装置。
【請求項９】　請求項８に記載の車両用プログラム開発支援装置において、
物理値に対応する浮動小数点数および浮動小数点数から変換された整数をそれぞれ適用して前記データフローチャートのシミュレーションを行い、浮動小数点数および整数のシミュレーション結果を出力するシミュレーション機能を有することを特徴とする車両用プログラム開発支援装置。
【請求項１０】　請求項９に記載の車両用プログラム開発支援装置において、
浮動小数点数および整数を適用したときのシミュレーション結果の相違を判定可能なように、整数のシミュレーション結果から浮動小数点数を逆算した結果を表示することを特徴とする車両用プログラム開発支援装置。
【請求項１１】　請求項１０に記載の車両用プログラム開発支援装置において、
前記データフローチャートのブロックシンボルに、浮動小数点数、整数、浮動小数点数から整数への整数変換条件、および該整数変換条件を用いて浮動小数点数から整数を逆算した結果、についての情報をもたせたことを特徴とする車両用プログラム開発支援装置。
【請求項１２】　請求項１１に記載の車両用プログラム開発支援装置において、
前記シミュレーション結果に基づいて前記整数変換条件を調整可能であることを特徴とする車両用プログラム開発支援装置。
【請求項１３】　請求項８～１２のいずれかに記載の車両用プログラム開発支援装置において、
前記ステートフローチャート上で同一階層の複数の前記データフローチャートを実行する順番を定義する優先付け機能を有することを特徴とする車両用プログラム開発支援装置。
【請求項１４】　請求項８～１３のいずれかに記載の車両用プログラム開発支援装置において、
前記データフローチャート内の所望のシンボル接続ラインを選択して所望のラベルを付けるラベリング機能を有し、

該当個所の変数名として前記ラベルを用いた前記車両用コードが生成されることを特徴とする車両用プログラム開発支援装置。

【請求項１５】　請求項８～１４のいずれかに記載の車両用プログラム開発支援装置において、

車両用コード生成の際に、前記データフローチャート内の複数のブロックシンボルに対応する複数の処理をグルーピングするグルーピング機能を有することを特徴とする車両用プログラム開発支援装置。

【請求項１６】　請求項１５に記載の車両用プログラム開発支援装置において、

グルーピングされるブロックシンボルの数を規定する所定のグルーピング制限条件に従ってグルーピングを行うことを特徴とする車両用プログラム開発支援装置。

【請求項１７】　請求項１５または１６のいずれかに記載の車両用プログラム開発支援装置において、

前記データフローチャート内の所望のシンボル接続ラインを選択して所望のラベルを付けるラベリング機能を有し、

前記ラベルを付けられた箇所が、グルーピングの区切りとされることを特徴とする車両用プログラム開発支援装置。

【請求項１８】　請求項８～１７のいずれかに記載の車両用プログラム開発支援装置において、

前記プログラムコード生成機能により、一般のＣコードを車両ＥＣＵに適合するように変形した車両用Ｃコードが生成されることを特徴とする車両用プログラム開発支援装置。

【請求項１９】　車両の制御仕様を表すデータフローチャートおよびステートフローチャートを作成するチャート作成機能と、

作成されたチャートに基づいて、車両ＥＣＵで処理される整数ロジックで構成された車両制御プログラムの車両用コードを生成するプログラムコード生成機能と、

をコンピュータに実現させるためプログラムを記録した、車両用プログラム開発支援に用いられるコンピュータ読取可能な記録媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、車両用プログラム(vehicle-use program)開発支援方法および装置に関し、特に、開発効率の向上および開発されるプログラムの信頼性の向上に関する。

【０００２】

【従来の技術】ビジュアルプログラミング技術を利用した制御システム設計支援統合環境を実現するための技術が進展している。このような背景の下、車両の制御システムの開発にも、シミュレーション機能およびプログラミング機能を備えたソフトウエアをインストールしたコンピュータが利用されるようになってきた。

【０００３】システム開発を行うユーザは、シミュレーション機能を利用して制御仕様書(specification)をコンピュータに入力する。コンピュータのディスプレイ上には、ユーザのアイデアを盛り込んだ仕様書が、データフローチャートおよびステートフローチャートといったかたちで書き表される。作成されたチャートに従ったシミュレーション計算が行われ、シミュレーション結果を参考にしてロジックのデバッグ作業が行われる。

【０００４】この種のシミュレーションソフトウエアは、作成されたチャートからプログラムコード、例えばＣコードを自動的に生成する機能をもつ。しかし、自動生成されるＣコードは、冗長で、容量が大きく、そのため、メモリの制約や実行速度に関する制約の厳しい車両制御にそのまま使うのには適していない。また、従来のコード生成方法では、コンピュータが決めた変数名がチャートの各部に割り付けられる結果、生成されたＣコードは可読性の低いものとなってしまう。

【０００５】そこで、ユーザは、データフローチャートおよびステートフローチャートを参考にして、車両ＥＣＵ(electric contol unit)に適した車両用Ｃコードを手作業で作成する。特に、一般のＣコードと異なり、整数ロジックで構成される専用コードが作成され、これにより、データビット数の観点からも少ないメモリ容量で高速処理が実現される。手作業で作られた車両用Ｃコードは実際に車載ＥＣＵにダウンロードされ、そしてデバッグ作業が行われる。

【０００６】

【発明が解決しようとする課題】従来は、上記のように、仕様シミュレーションとプログラムコーディングが全く別の作業であり、それぞれの段階で独立してデバッグ作業を行わなければならなかった。特に、車両用コードのデバッグ作業において不具合が見つかった場合、不具合原因がコーディングのミスにあるのか、不適当なオリジナル仕様書にあるのかを判定するのが困難であった。手作業のコーディングおよびデバッグ作業の困難さは、制御プログラムの信頼性保証にかかる工数を増大させ、さらには、開発期間を引き延ばし、開発コストの増大を招く可能性があった。

【０００７】本発明は上記課題に鑑みてなされたものであり、その目的は、信頼性の高い制御プログラムをより短期間で開発できるようにする車両用プログラム開発支援方法および装置を提供することにある。

【０００８】

【課題を解決するための手段】（１）上記目的を達成するため、本発明の車両用プログラム開発支援方法によれば、入力された制御仕様から車両用コードを生成する機能をもつプログラム生成装置を用いて、車両制御プログラムが生成される。生成された前記車両制御プログラムは車両ＥＣＵにダウンロードされる。そして、前記車両ＥＣＵに前記車両制御プログラムを実行させて、前記車

両制御プログラムのデバッグが行われる。このように、本発明によれば、制御仕様から直接的に生成された車両用コードが車両ＥＣＵにダウンロードされる。元の制御仕様と車両用コードが共通性をもつので、バグ発生が低減し、少ない作業量で効率よくデバッグ作業を行うことが可能となる。また、コーディング作業における開発者の手作業が大幅に削減される。

【０００９】好ましくは、前記デバッグステップのデバッグ作業は、前記車両ＥＣＵが前記車両制御プログラムを実行した結果を監視する前記プログラム生成装置上で行われる。制御仕様をもっているプログラム生成装置を利用することで、デバッグ作業が容易になる。

【００１０】（２）本発明の一態様の車両用プログラム開発支援装置は、車両の制御仕様を表すデータフローチャートおよびステートフローチャートを作成するチャート作成機能と、作成されたチャートに基づいて、車両ＥＣＵで処理される整数ロジックで構成された車両制御プログラムの車両用コードを生成するプログラムコード生成機能と、を有する。車両ＥＣＵでは、メモリサイズや実行速度の制約があるので整数ロジックを用いることが好適である。本態様では、データフローチャートおよびステートフローチャートから整数ロジックのコードを生成することで、車両用コードの自動生成ができる。

【００１１】好ましくは、本発明の一態様では、物理値に対応する浮動小数点数および浮動小数点数から変換された整数をそれぞれ適用して前記データフローチャートのシミュレーションを行い、浮動小数点数および整数のシミュレーション結果を出力するシミュレーション機能が設けられる。従来技術では、シミュレーション段階においては浮動小数点数が扱われ、コーディング段階では整数が扱われた。しかし、本発明によれば、シミュレーション段階で浮動小数点数と整数が処理され、両方の処理結果が出力されるので、仕様作成段階で整数ロジックの妥当性を容易に判定することができる。好ましくは、浮動小数点数および整数を適用したときのシミュレーション結果の相違を判定可能なように、整数のシミュレーション結果から浮動小数点数を逆算した結果を表示する。

【００１２】また、好ましくは、前記データフローチャートのブロックシンボルに、浮動小数点数、整数、浮動小数点数から整数への整数変換条件、および該整数変換条件を用いて浮動小数点数から整数を逆算した結果、についての情報をもたせる。ブロックシンボルのもつ整数情報を利用して車両用コードが自動生成される。

【００１３】（３）本発明の一態様では、前記ステートフローチャート上で同一階層の複数の前記データフローチャートを実行する順番を定義する優先付け機能が設けられる。これにより、無駄が無く適正な制御プログラムを確実に作成できる。

【００１４】また、本発明の一態様では、前記データフローチャート内の所望のシンボル接続ラインを選択して所望のラベルを付けるラベリング機能が設けられる。そして、該当個所の変数名として前記ラベルを用いた前記車両用コードが生成される。ユーザに分かりやすいラベルを付けることで、可読性の高い車両用コードが生成される。

【００１５】また、本発明の一態様では、車両用コード生成の際に、前記データフローチャート内の複数のブロックシンボルに対応する複数の処理をグルーピングするグルーピング機能が設けられる。好ましくは、グルーピングされるブロックシンボルの数を規定する所定のグルーピング制限条件に従ってグルーピングが行われる。また好ましくは、シンボル接続ラインにラベルを付けた箇所が、グルーピングの区切りとされる。このようなグルーピングにより、車両用コードの可読性を向上することができる。

【００１６】本発明の車両用コードは、例えば、一般のＣコードを車両ＥＣＵに適合するように変形した車両用Ｃコードであるが、ただしＣコードには限定されない。本発明の車両ＥＣＵは、車両に搭載される任意のＥＣＵであり、例えば、エンジンＥＣＵ、トランスミッションＥＣＵ、サスペンション制御ＥＣＵ、ブレーキ制御ＥＣＵである。ＥＣＵおよび制御対象の車載機器に応じて、プログラム開発に必要な制御仕様および車両仕様が異なることはもちろんである。

【００１７】

【発明の実施の形態】以下、本発明の好適な実施の形態（以下、実施形態という）について、図面を参照し説明する。

【００１８】図１を参照すると、プログラム生成装置１は、制御仕様の作成機能と、制御仕様のシミュレーション機能と、制御仕様から車両用Ｃコードを生成する機能を有する。

【００１９】プログラム生成装置１はコンピュータで構成され、図２に示すように、ＣＰＵ１０、ＲＯＭ１１、ＲＡＭ１２、キーボード１３、ポインティングデバイス１４、ディスプレイ１５、ハードディスク１６、通信回路１７、ＣＤ－ＲＯＭドライブ１８を有する。プログラム生成装置１の各種機能を実現するプログラムは、ＲＯＭ１１、ハードディスク１６あるいはＣＤ－ＲＯＭ１９に格納されており、ＣＰＵ１０により実行される。入出力装置は、キーボード１３、ポインティングデバイス１４、ディスプレイ１５には限定されない。また、ＣＤ－ＲＯＭ１９以外の任意のタイプの記録媒体、例えばＤＶＤが用いられてもよい。さらに、ハードディスク１６以外の任意のタイプの記憶装置が適用されてもよい。

【００２０】ユーザは、キーボード１３およびポインティングデバイス１４（マウス等）を操作し、自らのアイデアを盛り込んだ仕様書(specification)をディスプレイ１５に書き表す。ディスプレイ１５には、図３（ａ）

（ｂ）に示すデータフローチャート(data flowchart、ブロック線図)およびステートフローチャート（stateflowchart、state transition chart、状態遷移図)のかたちで制御仕様書が描かれる。データフローチャートが部分的なデータの流れを表し、ステートフローチャートが全体的な制御の流れを表す。

【００２１】また、ユーザは、キーボード１３およびポインティングデバイス１４を操作して車両仕様（諸元）を入力し、これにより、プログラム生成装置１内に仮想的な車両モデルが形成される。具体的には、車両モデルは、車両の種々の運動、動作等を記述した式の集合で構成される。

【００２２】ユーザの指示に応え、作成された仕様書（データフローチャートおよびステートフローチャート）のシミュレーションが行われる。図１に示すように、コンピュータ上で制御仕様のロジックが車両モデルを制御する。シミュレーション結果がディスプレイ１５に表示され、この段階で必要なデバッグ作業が行われる。デバッグ作業を行いながら、仕様書の修正および改良が行われ、仕様書が完成される。

【００２３】仕様書が完成すると、ユーザの指示に従い、仕様書から車両用Ｃコードが生成される。すなわち、データフローチャートおよびステートフローチャートに対応するＣコードを書き込んだファイルが形成される。車両用Ｃコードは整数ロジックを採用しており、一般のＣコードと比較して少ないメモリサイズで高速処理を実現可能に構成されている（後述）。生成された車両用Ｃコードは、実際の車両ＥＣＵ２、例えばエンジンＥＣＵにダウンロードされる。

【００２４】また、プログラム生成装置１から車両モデル装置３へは、車両モデルがダウンロードされる。ＤＳＰを含む車両モデル装置３は、高速の割込処理等により、車両ＥＣＵ２にとっては見かけ上は実際の車両と同様に動作する。開発期間の短縮と効率のよい開発のためには、実際の車両を使ったテストの前にこのようなシミュレーションを行うことが効果的である。

【００２５】車両モデル装置３が車両ＥＣＵ２に接続され、車両ＥＣＵ２にダウンロードした車両用Ｃコードが実行される。これにより、現実に近い環境でのシミュレーションが行われる。車両ＥＣＵ２および車両モデル装置３の動作がプログラム生成装置１により通信回路１７を使ってモニタされ、そして、プログラム生成装置１上でデバッグ作業が行われる。例えば、不適当な制御動作が発見されると、その原因がデータフローチャートおよびステートフローチャートを使って調べられる。コンピュータ上で不具合箇所の修正が行われ、修正後のロジックが検証される。

【００２６】デバッグ作業の終了後、車両ＥＣＵ２は車両４に搭載され、実車テスト(actual test)が行われる。ここでは、実際の車両の非線形要因等に起因する修正必要箇所が検出され、最終的なプログラム修正作業が行われる。

【００２７】以下、プログラム生成装置１についてより詳細に説明する。

【００２８】［制御仕様の入力・作成機能（整数ロジック）］従来は、制御仕様を入力する段階では、物理値をそのまま表す浮動小数点数(floating point number)を処理するチャートが作成される。そして、浮動小数点数を計算したシミュレーション結果に基づいて、制御仕様の妥当性が判断される。従来機能によりチャートから自動的に生成されるＣコードも浮動小数点数を含む。

【００２９】しかし、車両制御では、低コストおよび処理速度の観点から、ＥＣＵのＣＰＵの能力を最大限に引き出すことが要求される。高速回転するエンジンなどの機器の制御においては、この要求が顕著である。このような要求に応えるため、車両ＥＣＵ用のＣコードに整数ロジック（固定小数点数(fixed point number)）を適用し、一般Ｃコードとはデータビット数を異ならせることが好適である。この点が、車両用Ｃコードと自動生成される一般Ｃコードとの大きな相違点であり、開発者による手作業での車両用Ｃコード作成を余儀なくする理由であった。

【００３０】本実施形態では、後述する整数ブロックを導入することにより、制御仕様の作成段階から整数ロジックを考慮できるようにし、そして、制御仕様から整数ロジックの車両用Ｃコードを自動生成可能にする。

【００３１】（１）浮動小数点数から整数への変換

浮動小数点数から整数への変換は、下式(1)に従って行われる。；

【数１】

$$x\_int = int(x\_float - OFFSET) / SLOPE \quad \cdots (1)$$

ここで、x_floatは浮動小数点数であり、x_intは整数であり、OFFSET（またはｂｉａｓ）およびSLOPE（またはLSB）は整数変換条件である。

【００３２】（２）整数ロジックの概要

図４には、車両ＥＣＵにおける整数ロジックの概念が示されている。センサ２１は、検出した物理量ｐに応じた電圧信号ｖをＡ／Ｄ変換器２２に送る。Ａ／Ｄ変換器２２は電圧信号ｖを整数データＰに変換し、車両ＥＣＵ２に送る。車両ＥＣＵ２は、整数ロジックを用いて、検出信号の整数データＰから制御パラメータの整数データＱを得る。整数データは、例えば、１６ｂｉｔの符号無し整数(unsigned integer)である。整数データＱは制御信号に変換され、制御対象のアクチュエータ２３に出力される。そして、アクチュエータ２３が動作して物理量ｑが生じる。

【００３３】図４において、整数ロジックは、物理値を対象とするオリジナルの制御ロジックと同等の結果をもたらさなければならない。車両ＥＣＵ２の入出力部のＳ

ＬＯＰＥ、ＯＦＦＳＥＴは、それぞれｌｐ、ｌｑ、ｏｐ、ｏｑである。整数ロジックには実際には複数段階の演算処理が設けられ、中間過程の幾つものＳＬＯＰＥ、ＯＦＦＳＥＴが設けられる。これらのＳＬＯＰＥ、ＯＦＦＳＥＴは、メモリサイズおよび実行時間の観点から整数ロジックを最適化するように設定されるべきである。

【００３４】例えば、ユーザが作成した制御仕様（ロジック）の一部に、図５に示すような伝達関数があったとする。整数ロジックの構成にあたり、伝達関数の７つの項のすべてについて同じＳＬＯＰＥを使ったとする。第１項と第７項では係数が大幅に違うため、第１項に適したＳＬＯＰＥを採用すると、第７項の数値は常にゼロになってしまい得る。

【００３５】一般的に、小さすぎるＳＬＯＰＥはオーバーフローの原因になる。また、大きすぎるＳＬＯＰＥは結果的に一部のセンサの入力の無視を招くことになる。さらに、不適当なＳＬＯＰＥ設定が原因で制御ロジックが冗長成分を含むこともあり得る。このようなことを回避するため、適切な整数変換条件を容易に設定可能にすることが求められる。

【００３６】（３）整数ブロック

データフローチャート上で適切な整数ロジックを構成し、適切な整数変換条件を容易に設定できるようにするために、本実施形態では、その特徴として図６の整数ブロック(Integer Block)を導入する。整数ブロックは、データフローチャート内の一つのブロックシンボルである。ユーザは、整数ブロックのシンボルをデータフローチャートに書き込むことができる。

【００３７】整数ブロックは、図６に示すように、(1)ｉｎｔｅｇｅｒ（整数）、(2)ＳＬＯＰＥ（LSB）、(3)ＯＦＦＳＥＴ（ｂｉａｓ）、(4)ｆｌｏａｔ（浮動小数点数）、(5)Ｄａｔａ　Ｔｙｐｅ、(6)“ｉｎｔｅｇｅｒ×ＳＬＯＰＥ＋ＯＦＦＳＥＴ”の６種の情報をもつ。（６）は、ｉｎｔｅｇｅｒからｆｌｏａｔを逆算した結果（逆算ｆｌｏａｔ）である。

【００３８】図７および図８を参照し、かけ算を例にして、整数ブロックの機能を説明する。図７のデータフローチャートにおいて、かけ算ブロックには、入力データ１、２とともに、それぞれ、ＳＬＯＰＥおよびＯＦＦＳＥＴが入力される。かけ算ブロックは、これらのデータから上記の６つの情報を求めて出力する。

【００３９】図８には、かけ算ブロックの処理が式で示されている。入力値ｘ、ｙがそれぞれｏｆｆｓｅｔ１、２およびｓｌｏｐｅ１、２を用いて整数に変換され、これら２つの整数の積が出力される（ｉｎｔｅｇｅｒ出力）。一方、ｆｌｏａｔ出力は、ｘｙである。第１式をｘｙと比較すると、出力側のＳＬＯＰＥおよびＯＦＦＳＥＴが分かる（図８、第２式および第３式）。さらに、逆算ｆｌｏａｔ（＝ｉｎｔｅｇｅｒ×$SLOPE_{out}$＋$OFFSET_{out}$）が出力される。

【００４０】ユーザは、図７の表示を見て、シミュレーション結果を知ることができる。ｆｌｏａｔを見ることで、基本の制御ロジックが正しいか否かが分かる。さらに、ｆｌｏａｔと逆算されたｆｌｏａｔが比較される。両者の差（０．１）が許容範囲内であれば、整数ロジックが適当であることが分かる。一方、両者の差が許容範囲外であれば、整数ロジック、すなわち整数変換条件であるｓｌｏｐｅ１、２の設定が不適当であることが分かる。そこで、適当な結果が得られるまでｓｌｏｐｅ１、２が調整される。

【００４１】図７および図８には、制御仕様の一部のみが例示されている。しかし、実際の制御仕様は、連結された一連の多数のブロックで構成される。本実施形態によれば、制御過程の各ブロックでＳＬＯＰＥの検証と調整を行うことができる。例えばあるブロックでｆｌｏａｔとｉｎｔｅｇｅｒのずれが大きければ、着目ブロック以前のいずれかのブロックの整数変換条件が調整される。

【００４２】従って、制御過程の全体において、ｆｌｏａｔとｉｎｔｅｇｅｒが実質的に同等の意味をもつことができる。言い換えれば、後段のブロックに行くに従ってｆｌｏａｔとｉｎｔｅｇｅｒとのずれが拡大するというような事態の発生が回避される。また、不適当なＳＬＯＰＥの設定によってあるセンサの入力が無視されるというような事態の発生も回避される。

【００４３】図８では、その第３式に示されるように、出力側のＯＦＦＳＥＴが、入力値ｘ、ｙを含んでいる。この影響が無視できない場合には、ユーザは、図９の変形処理を選択できる。図９では、その第１式に示されるように、かけ算の前に整数に“ｏｆｆｓｅｔ／ｓｌｏｐｅ”を足すことにより、出力側のｏｆｆｓｅｔの項がゼロになる。

【００４４】また、上記説明では、かけ算を例にして整数ブロックの機能を述べたが、他の種類の演算に関しても同様の機能が設けられる。図１０～図１６は、整数ロジックをサポートする各種の整数ブロックを示している。なお、図１０に示すように、このツールでは、整数ブロックに情報ｙ（５）～ｙ（８）が付加されているが、基本的な情報ｙ（１）～ｙ（４）は上記と同様である。また、図１０～図１６では、ＬＳＢがｓｌｏｐｅに相当している。

【００４５】以上のように、本実施形態では、整数ブロックの導入により、チャート形式の制御仕様書が、整数ロジックを表現する。理論的な制御ロジック（実数演算）と整数ロジック（整数演算）とを比較しながら適切な整数変換条件を設定できるので、バグの少ない適正な整数ロジックを容易に作成できる。作成された制御仕様書は、それ自体がコンピュータ上でシミュレーション可能である。さらに、制御仕様書の整数ロジック部分を抽出してＣコードを生成すれば、車両用Ｃコードを自動的

に生成できる。制御仕様書と車両用Ｃコードが共通性をもつので、コード生成段階でのバグ発生が低減し、また、デバッグ等のプログラム検証も容易になる。

【００４６】［優先付け機能（演算順序の規定）］複数のモジュールまたは演算式の順序を規定する場合、一般には、図１７に示すように、各モジュールまたは演算式がトリガー付きサブシステム（TriggeredSubsystem)とされ、それらのサブシステムに対してステートフローチャートからファンクションコール(function-call)が行われる。

【００４７】本実施形態では、さらに、図１８の優先付け機能が設けられている。ユーザは、データフローチャートに優先順位を表す番号を付ける。各モジュールまたは演算式は、優先付けされたサブシステム（Prioritized Subsystem)となる(1_Prior_Subsystem、2_Prior_Subsystem)。演算は、サブシステムの名前の順に行われる。

【００４８】この優先付け機能によれば、同一階層のデータフローチャートの演算順序の規定が容易にできるという利点が得られる。また、図１９に示すように、生成されるＣコードが従来より簡単になるという利点があり、これは可読性およびメモリ容量の面から好適である。

【００４９】［Ｃコード生成（ラベリング）］Ｃコード生成は、ユーザの指示に応えて行われる。前述のように、整数ブロックの適用により、データフローチャートの各ブロックが整数情報をもつ。従って、一連のブロック群を整数演算で辿ることができる。整数演算を書き出すことにより、整数ロジックで構成された車両用Ｃコードが生成される。具体的には、図７の例を参照すると、チャートの段階でかけ算ブロックに入出力される情報群の中から、整数演算（２つの整数の入力と１つの整数の出力）が抽出される。

【００５０】既存のソフトウエア製品を変形して利用する場合には、チャートから一般Ｃコードを自動生成するツールと、Ｃコードを書き込む雛形ツールとを用意し、そして、整数ロジックに適するように雛形を変形すればよい。このとき、一般Ｃコードにはない専用コマンド、および一般Ｃコードを変形した専用コマンド等に関する処置も施される。

【００５１】図２０は、データフローチャートから生成されるＣコードの一例を示している。通常のＣコード生成方法では、図２０に示すように、コンピュータが決めた変数名が各部に割り付けられる。そのため、生成されたＣコードは、人間にとっては非常に読みづらいものになる。

【００５２】このように一般的なＣコード自動生成方法では、可読性はあまり考慮されていない。主としてＣコードとチャートの一致が求められているからである。しかし、車両システム開発では、実際のＥＣＵを使ったテストで動作検証が行われる。このとき、開発者がＣコードを読みとらなければならない機会が多い。従って、車両用Ｃコードの自動生成では、可読性が高いことが求められる。

【００５３】本実施形態では、可読性向上のために、図２１に示すラベリング機能が設けられている。ユーザは、データフローチャート内のブロックシンボルの接続ラインをクリックする。選択した接続ラインに所望のラベルが付けられる。図２１では、例として、ラベル“t_x” “t_y” “t_z” “t_xyz”が付けられているが、実際には、該当するモジュール名やセンサ名などの分かりやすいラベルを付けることが好適である。Ｃコード生成の際には、ラベルが付けられた箇所の変数名として該当ラベルが使われる。その結果、図２１に示すように、読みやすいＣコードが生成される。

【００５４】なお、図２１の例ではすべての接続ラインにラベルが付けられている。しかし、ユーザは、Ｃコードを読みやすくするために適当な所望のラインにラベルを付ければよい。ラベリングされないラインについては、コンピュータが割り付けた変数名がそのまま使用される。

【００５５】［Ｃコード生成におけるグループ化］（１）通常のＣコード生成では、データフローチャートの一つ一つのブロックに個別の変数名が付けられる。そして、各ブロックに対応して、一つの変数を導き出す式が生成される。そのため、生成されたＣコードは、多数の変数と式を含み、冗長で読みづらい。

【００５６】そこで、本実施形態では、可読性の向上のためにグルーピング機能が設けられている。図２２を参照すると、データフローチャートでは、入出力の間に２つのブロックがある。グループ化が行われていない左側のＣコードでは、ブロック数に応じた２つの式が生成されている。一方、グループ化された右側のＣコードでは、２つのブロックの演算がまとめられ、一つの式で表されている。人間がＣコードを読む上では不要な変数名が削減されており、従って、読みやすいＣコードが生成されている。

【００５７】ただし、多数のブロックをグルーピングすると、一つの式が長くなりすぎて、かえって可読性を低下させる可能性がある。そこで、一つのグループに含まれるブロックの数を規定する制限条件を設けることが好適である。ここでは、ブロックの数の最大値を２つとする。従って、Ｃコードの一つの式は、最大２段階の演算を行う。

【００５８】図２３および図２４を参照し、グルーピング処理を説明する。Ｃコード生成の際、データフローチャートの各ブロックにＩＤが付けられ、そのＩＤが変数名として使われる。ＩＤとユーザがつけたラベルとを関連づける変数名（信号名）データベース（図２３（ａ））が作成される。ラベルが付けられていない場合には、ＩＤがそのまま使用される。

【００５９】図２４において、まず、入力信号を調べる関数によって、着目するブロックの入力信号のＩＤが調べられる（Ｓ１１）。そして、式のデータベース（図２３（ｂ））を参照して、各入力信号について、グルーピング条件（最大２回）が成立しているか否かが判断される（Ｓ１２）。条件未成立（２回未満）であれば、数式のデータベースが検索される（Ｓ１３）。そして、入力ＩＤに対応する式が採用される。一方、条件が成立すれば、変数名のデータベースが検索され（Ｓ１４）、該当するラベルが変数名として採用される。ラベルがなければＩＤが採用される。

【００６０】Ｓ１３の式またはＳ１４の変数名を用いて、着目ブロックの式が作成される（Ｓ１５）。作成した式に関してグルーピング条件が成立したか否かが判断される（Ｓ１６）。条件未成立であれば、次のブロックでの処理のために、式のデータベースに、Ｓ１５で作った式が登録される（Ｓ１７）。グルーピング条件が成立すれば、Ｓ１５で作った式がファイルに書き出される（Ｓ１８）。図２２では、２つのブロックをまとめた式が出力される。

【００６１】図２５を参照して、上記のグルーピング処理の具体例を説明する。入力信号はｓ１およびｓ２である。入力信号ｓ１について式のデータベースを検索すると、式（ｘ＋１）が得られる。この式のグルーピング回数はまだ１回である（条件未成立）。そこで、式（ｘ＋１）を入力信号として採用する。次に、入力信号ｓ２についての検索を行うと、入力信号ｓ２のグルーピング回数は２回であることが分かる。そこで、変数のデータベースが検索され、ラベルｘ２が入力信号として採用される。従って、着目しているブロック内の演算は、（ｘ＋１）×ｘ２である（かけ算）。

【００６２】次に、変数名のデータベースから出力信号名t_xが検索される。入力信号ｓ１に関して２回目のグルーピングを行っているので、グルーピング条件が成立している。そこで、t_x=（ｘ＋１）×ｘ２が出力される。もしこの段階でまだグルーピング条件が成立していなければ、作成された式はデータベースに登録される。

【００６３】（２）グルーピング処理では、さらに、チャート内でラベルが付けられた場所は、必ずグループの区切りとされる。ユーザが付けたラベルはすべてＣコードの演算式に含まれる。ユーザがチャート内の適当な場所にラベルを付ければ、チャートの構造に対応する構成をもつＣコードが生成される。このような処理により、自動生成されるＣコードがさらに読みやすくなる。

【００６４】例えば、ユーザが、データフローチャートの最初と最後にラベルを付けたとする。この場合、最初と最後の変数を明瞭に示し、かつ、処理過程を適当な間隔で区切られた、可読性の高いＣコードが自動生成される。

【００６５】以上より、本実施形態によれば、Ｃコード生成の際に適切なグルーピングを行うことにより、自動生成されるＣコードの可読性をさらに高めることが可能となる。また、変数名を減らしてメモリを節約することができる。

【００６６】［車両ＥＣＵへのダウンロード］図１を参照して述べたように、プログラム生成装置１は、ユーザの指示に応え、自動生成した車両用Ｃコードを車両ＥＣＵ２にダウンロードする。車両ＥＣＵ２は車両用Ｃコードを実行して車両モデル装置３（ＤＳＰ）を制御する。プログラム生成装置１は、制御の実行の様子をモニタし、ディスプレイ上に適当なかたちで表示する。図２６は、車両ＥＣＵ２にエンジンＥＣＵを適用した場合のモニタ画面例を示している。ユーザは、プログラム生成装置１を操作して、車両用Ｃコードの動作検証およびデバッグ作業を行う。

【００６７】本実施形態では、バグの少ない適正な車両用Ｃコードが生成されている。車両用Ｃコードと元の制御ロジックが共通している。さらに、車両用Ｃコードの可読性が高い。従って、デバッグ作業を効率よく容易に行うことができる。

【００６８】

【発明の効果】以上に説明したように、本発明によれば、ユーザが入力した制御仕様に対応する車両用コードが自動的に生成されるので、コーディング作業の工数が大幅に削減される。入力された制御仕様は、それ自体がコンピュータ上でシミュレーション可能なものであり、かつ、車両用コードに変換して車両ＥＣＵでも使用可能なものであり、従って、デバッグ作業の工数の大幅削減が可能となる。その結果、信頼性保証のためにかかる工数が削減され、さらに、開発期間の短縮および開発コストの低減が可能になる。特に、車両制御プログラムの肥大化と複雑化という背景の下では、仕様書作成からロジックデバッグまでを支援する本発明の利点が顕著に得られる。

【００６９】また、本発明によれば、ラベリングやグルーピングを行う適切なＣコード生成処理により、可読性の高い車両用コードが自動生成される。これにより、デバッグ作業等の工数をさらに削減することができる。

【図面の簡単な説明】

【図１】　本発明の実施形態の全体構成を示す図である。

【図２】　プログラム生成装置の構成を示すブロック図である。

【図３】　制御仕様を表すデータフローチャートおよびステートフローチャートを示す図である。

【図４】　車両ＥＣＵにおける整数ロジックの概念を示す図である。

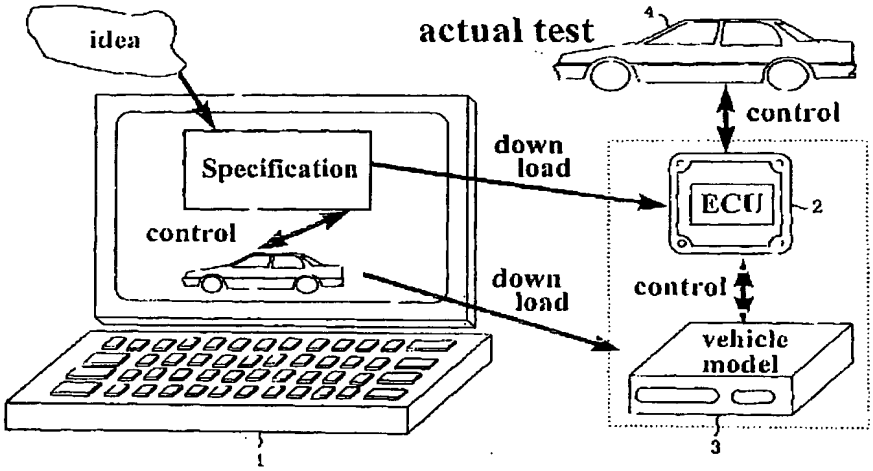
【図５】　整数ロジックの一部の例を示す図である。

【図６】　整数ブロックの構成を示す図である。

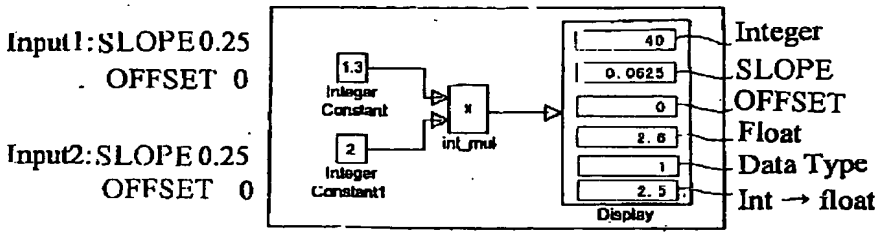
【図７】　整数ブロックをかけ算に適用した例を示す図

である。
【図８】　図７のかけ算ブロックの処理を示す図である。
【図９】　図８の処理の変形例を示す図である。
【図１０】　整数ロジックのサポートツールを示す図である。
【図１１】　整数ロジックのサポートツールを示す図である。
【図１２】　整数ロジックのサポートツールを示す図である。
【図１３】　整数ロジックのサポートツールを示す図である。
【図１４】　整数ロジックのサポートツールを示す図である。
【図１５】　整数ロジックのサポートツールを示す図である。
【図１６】　整数ロジックのサポートツールを示す図である。
【図１７】　演算順序の規定方法を示す図である。
【図１８】　演算順序の規定方法であって本実施形態の優先付け機能を示す図である。
【図１９】　図１７および図１８の結果として生成されるＣコードを示す図である。
【図２０】　データフローチャートから生成されるＣコードの例を示す図である。
【図２１】　図２０と同様のデータフローチャートから生成されるＣコードであって、本実施形態のラベリングが行われたＣコードを示す図である。
【図２２】　Ｃコード生成におけるグルーピング処理を示す図である。
【図２３】　Ｃコード生成過程で作成される変数名データベースおよび式のデータベースを示す図である。
【図２４】　図２３のデータベースを利用したグルーピング処理を示す図である。
【図２５】　グルーピング処理の具体例を示す図である。
【図２６】　プログラム生成装置がエンジン制御をモニタするときの表示画面例を示す図である。
【符号の説明】
１　プログラム生成装置、２　車両ＥＣＵ、３　車両モデル装置、１０　ＣＰＵ、１１　ＲＯＭ、１２　ＲＡＭ、１３　キーボード、１４　ポインティングデバイス、１５　ディスプレイ、１６　ハードディスク、１７　通信回路、１９ＣＤ－ＲＯＭ。

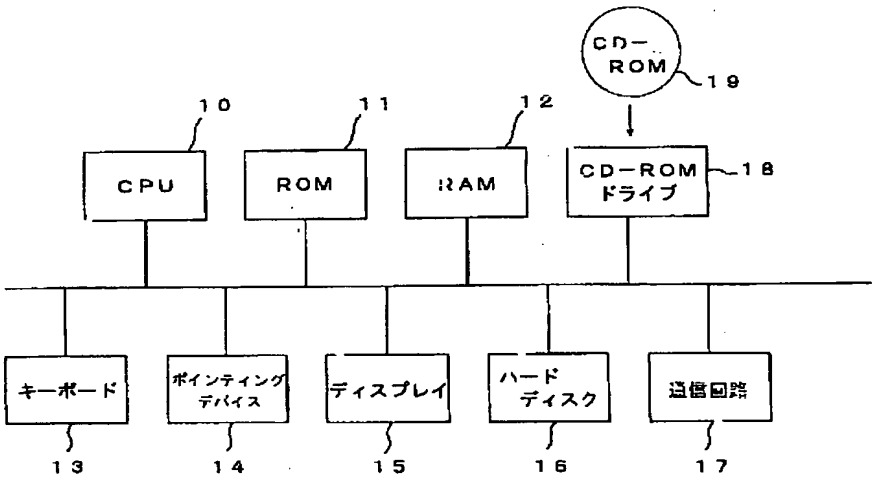
【図１】



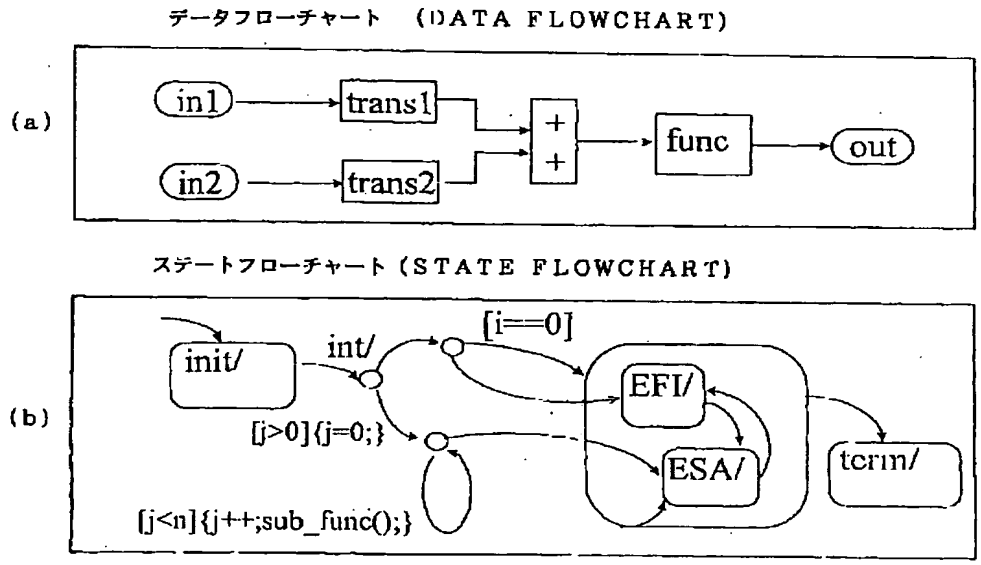
【図７】



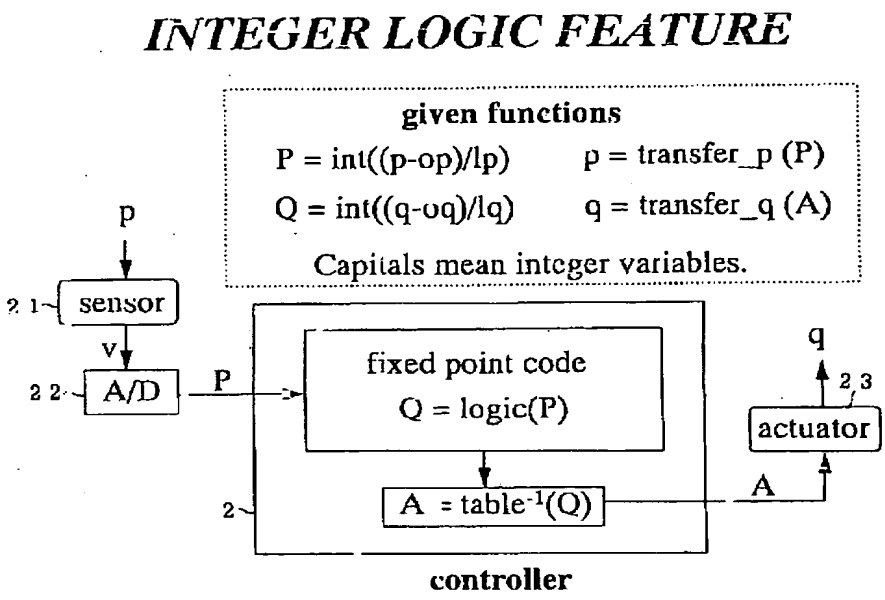
【図2】


CD-ROM
19
10
11
12
CPU
ROM
RAM
CD-ROMドライブ
18
キーボード
ポインティングデバイス
ディスプレイ
ハードディスク
通信回路
13
14
15
16
17

【図3】


データフローチャート (DATA FLOWCHART)
(a)
in1
in2
trans1
trans2
+
+
func
out
ステートフローチャート (STATE FLOWCHART)
(b)
init/
int/
[i==0]
[j>0]{j=0;}
[j<n]{j++;sub_func();}
EFI/
ESA/
term/

【図4】

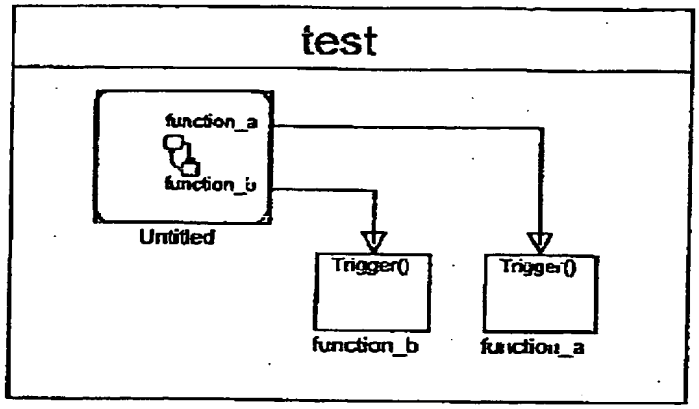


【図5】

y(k) = 0.7888 y(k-1) +
0.1784 y(k-2) -
0.1000 y(k-3) -
0.0010 u(k) +
0.0150 u(k-1) -
0.0040 u(k-2) -
0.0020 u(k-3)

example of
floating point formula

【図17】



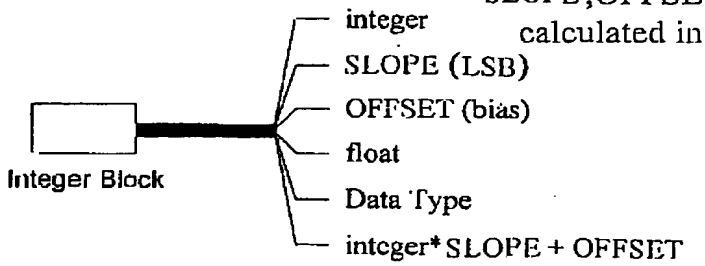
【図6】

Integer Block's Signal

Translation formula
float → integer

x_int = (x_float - OFFSET)/SLOPE

SLOPE,OFFSET is also calculated in the integer block



【図8】

整数ブロックのかけ算（1）

multiply

$$\frac{\text{input1}}{\text{slope1}}(x - \text{offset1}) \times \frac{\text{input2}}{\text{slope2}}(y - \text{offset2})$$

$$= \frac{\text{Output}}{\text{slope1*slope2}}(xy - \text{offset1}*y - x*\text{offset2} + \text{offset1}*\text{offset2})$$

Output SLOPE : slope1*slope2
Output OFFSET : offset1*y + x*offset2 - offset1*offset2

【図9】

整数ブロックのかけ算（2）

multiply

$$\left(\frac{(x - offset1)}{slope1}_{X_{int}} + \frac{offest1}{slope1}\right) \times \left(\frac{(y - offset2)}{slope2}_{Y_{int}} + \frac{offset2}{slope2}\right) = \frac{xy}{slope1*slope2}_{Output}$$

specifies SLOPE : slope_o → output
specified OFFSET : offset_o→ output
$OFF1_{int}$: offset1/slope1, $OFF2_{int}$: offset2/ slope2
multiplied integer value :

$$((X_{int} + OFF1_{int})*(Y_{int}+OFF2_{int})*slope1*slope2 - offset\_o)/slope\_o$$

【図10】

INTEGER LOGIC SUPPORT (1)

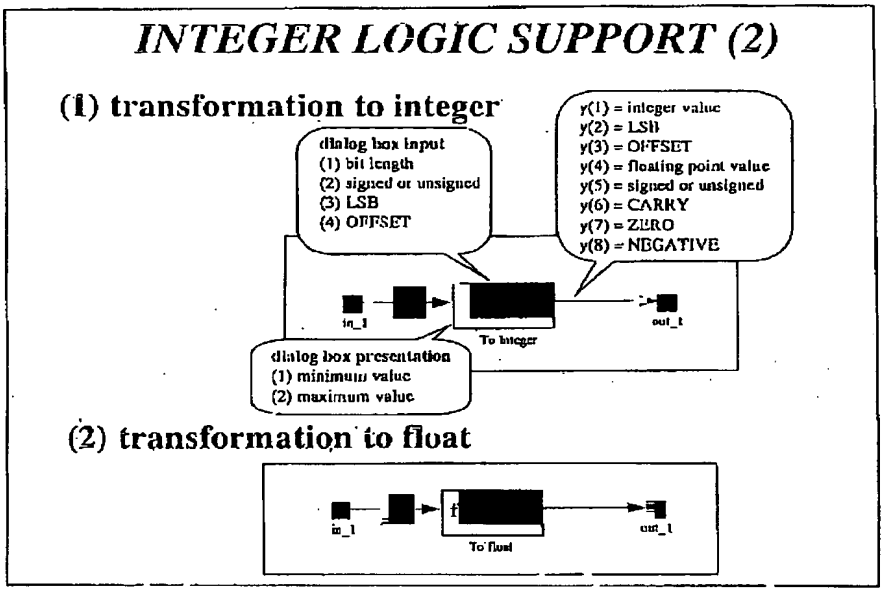
(1) All integer lines are vectors composed of

y(1) = integer value
y(2) = LSB
y(3) = OFFSET
y(4) = floating point value
y(5) = signed or unsigned
y(6) = CARRY
y(7) = ZERO
y(8) = NEGATIVE

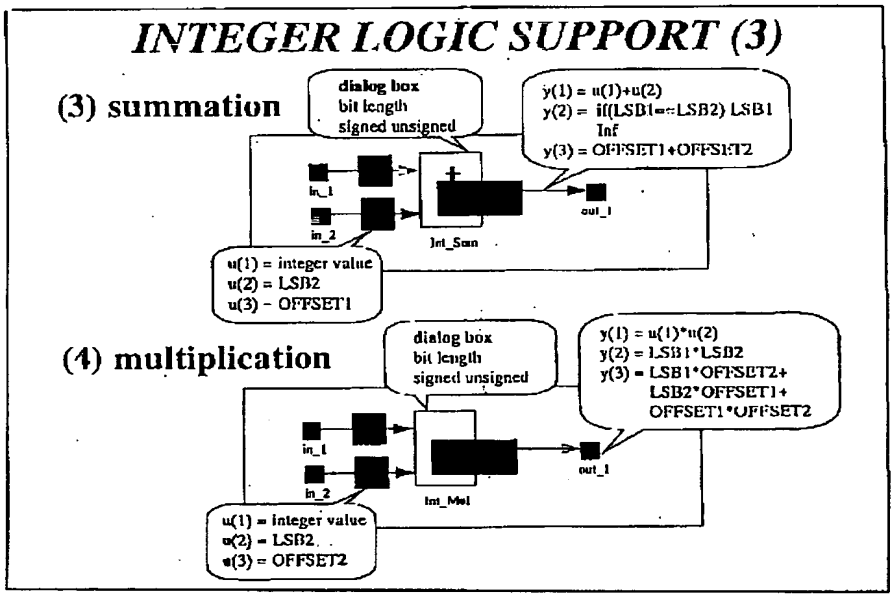
(2) The integer logic support tool has the folowing blocks.

(1) transformation to integer
(2) transformation to float
(3) summation
(4) multiplication
(5) division
(6) residual
(7) shift
(8) one D table look up
(9) two D table look up
(10) split
(11) unit delay
(12) integer scope

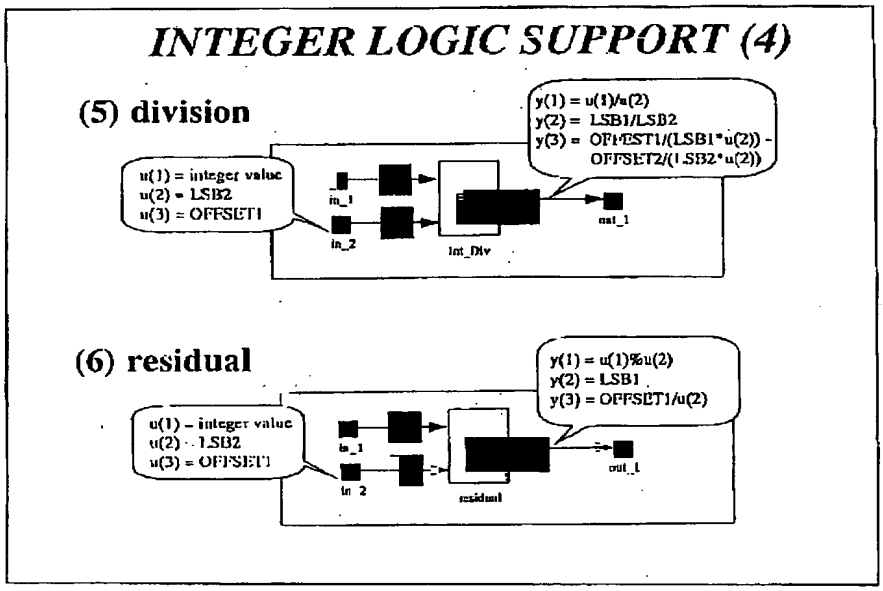
【図11】


INTEGER LOGIC SUPPORT (2)
(1) transformation to integer
dialog box input
(1) bit length
(2) signed or unsigned
(3) LSB
(4) OFFSET
y(1) = integer value
y(2) = LSB
y(3) = OFFSET
y(4) = floating point value
y(5) = signed or unsigned
y(6) = CARRY
y(7) = ZERO
y(8) = NEGATIVE
in_1
To Integer
out_1
dialog box presentation
(1) minimum value
(2) maximum value
(2) transformation to float
in_1
f
To float
out_1

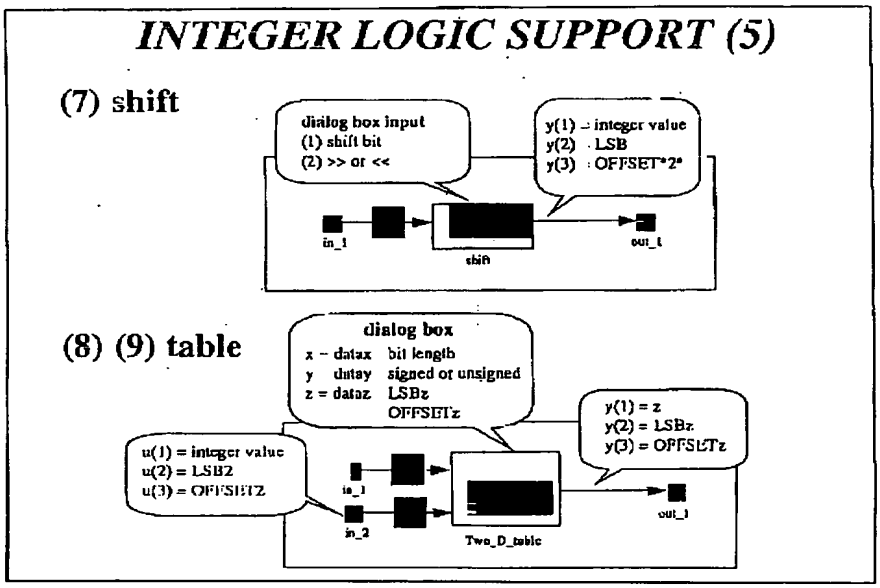
【図12】


INTEGER LOGIC SUPPORT (3)
(3) summation
dialog box
bit length
signed unsigned
y(1) = u(1)+u(2)
y(2) = if(LSB1==LSB2) LSB1
Inf
y(3) = OFFSET1+OFFSET2
in_1
in_2
Int_Sum
out_1
u(1) = integer value
u(2) = LSB2
u(3) = OFFSET1
(4) multiplication
dialog box
bit length
signed unsigned
y(1) = u(1)*u(2)
y(2) = LSB1*LSB2
y(3) = LSB1*OFFSET2+
LSB2*OFFSET1+
OFFSET1*OFFSET2
in_1
in_2
Int_Mul
out_1
u(1) = integer value
u(2) = LSB2
u(3) = OFFSET2

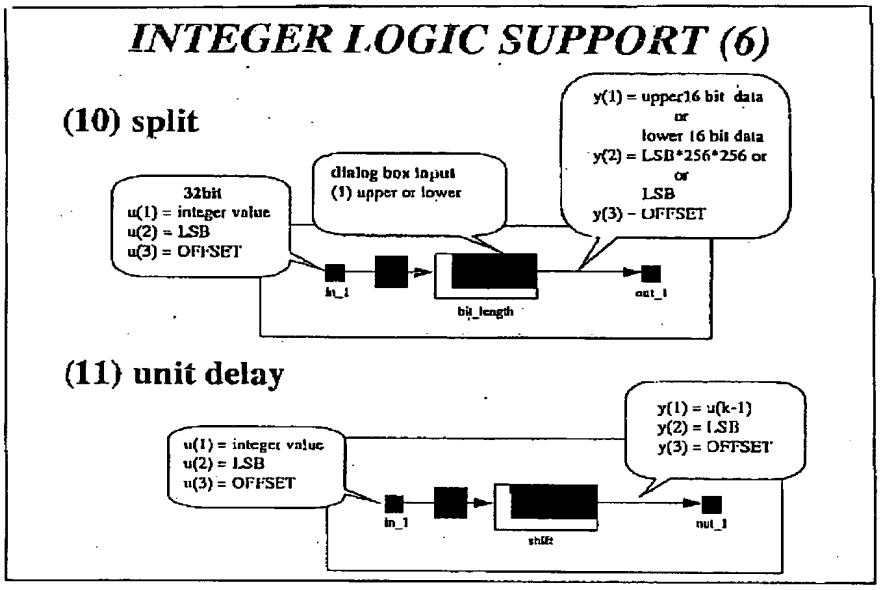
【図13】


INTEGER LOGIC SUPPORT (4)
(5) division
y(1) = u(1)/u(2)
y(2) = LSB1/LSB2
y(3) = OFFSET1/(LSB1*u(2)) -
OFFSET2/(LSB2*u(2))
u(1) = integer value
u(2) = LSB2
u(3) = OFFSET1
in_1
in_2
Int_Div
out_1
(6) residual
y(1) = u(1)%u(2)
y(2) = LSB1
y(3) = OFFSET1/u(2)
u(1) = integer value
u(2) = LSB2
u(3) = OFFSET1
in_1
in_2
residual
out_1

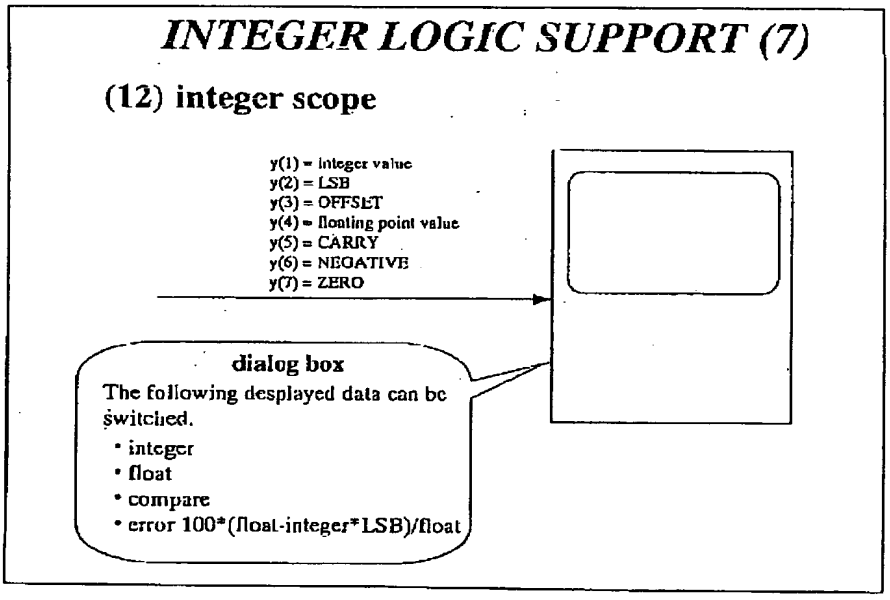
【図14】


INTEGER LOGIC SUPPORT (5)
(7) shift
dialog box input
(1) shift bit
(2) >> or <<
y(1) = integer value
y(2) = LSB
y(3) = OFFSET*2^
in_1
shift
out_1
(8) (9) table
dialog box
x = datax bit length
y = datay signed or unsigned
z = dataz LSBz
OFFSETz
y(1) = z
y(2) = LSBz
y(3) = OFFSETz
u(1) = integer value
u(2) = LSB2
u(3) = OFFSETZ
in_1
in_2
Two_D_table
out_1

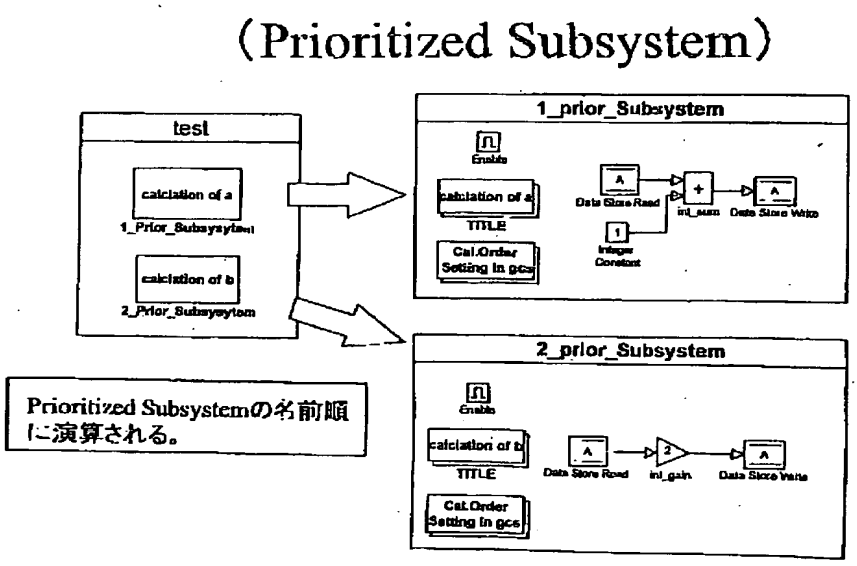
【図15】


INTEGER LOGIC SUPPORT (6)
(10) split
32bit
u(1) = integer value
u(2) = LSB
u(3) = OFFSET
dialog box input
(1) upper or lower
y(1) = upper16 bit data
or
lower 16 bit data
y(2) = LSB*256*256 or
or
LSB
y(3) = OFFSET
in_1
bit_length
out_1
(11) unit delay
u(1) = integer value
u(2) = LSB
u(3) = OFFSET
y(1) = u(k-1)
y(2) = LSB
y(3) = OFFSET
in_1
shift
out_1

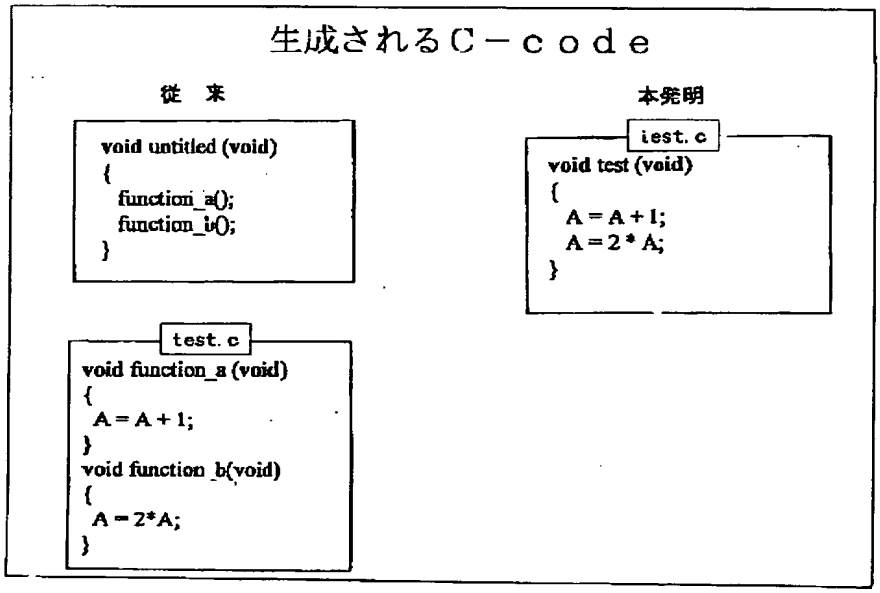
【図16】


INTEGER LOGIC SUPPORT (7)
(12) integer scope
y(1) = integer value
y(2) = LSB
y(3) = OFFSET
y(4) = floating point value
y(5) = CARRY
y(6) = NEGATIVE
y(7) = ZERO
dialog box
The following desplayed data can be switched.
• integer
• float
• compare
• error 100*(float-integer*LSB)/float

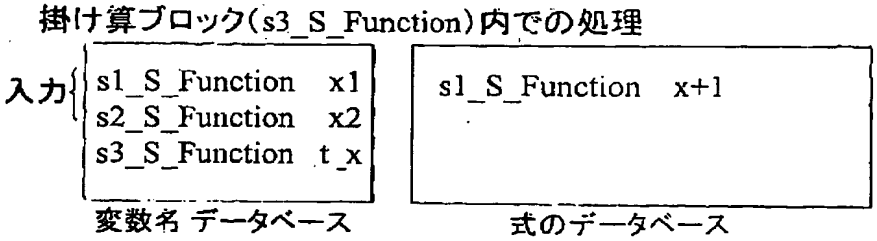
【図18】


（Prioritized Subsystem）
test
calclation of a
1_Prior_Subsystem
calclation of b
2_Prior_Subsystem
Prioritized Subsystemの名前順に演算される。
1_prior_Subsystem
Enable
calclation of a
TITLE
Cal.Order Setting in gcs
Data Store Read
int_sum
Data Store Write
Integer Constant
2_prior_Subsystem
Enable
calclation of b
TITLE
Cal.Order Setting in gcs
Data Store Read
int_gain
Data Store Write

【図19】

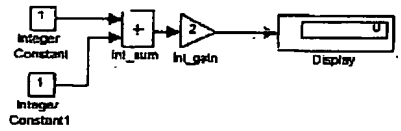

生成されるC－ｃｏｄｅ
従　来
本発明
void untitled (void)
{
function_a();
function_b();
}
test.c
void function_a (void)
{
A = A + 1;
}
void function_b(void)
{
A = 2*A;
}
test.c
void test (void)
{
A = A + 1;
A = 2 * A;
}

【図25】


掛け算ブロック(s3_S_Function)内での処理
入力
s1_S_Function x1
s2_S_Function x2
s3_S_Function t_x
変数名 データベース
s1_S_Function x+1
式のデータベース

【図20】

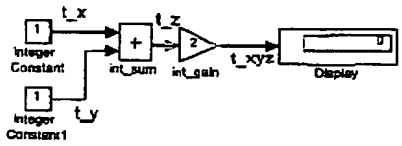
C-code生成




```
void untitled(void)
{
 s16 s1_S_Function;
 s16 s2_S_Function;
 s16 s4_S_Function;
 s16 s3_S_Function;

 /* int_sum : s4_S_Function */
 s4_S_Function = s1_S_Function+s2_S_Function;

 /* int_gain : s3_S_Function */
 s3_S_Function = (s16)(2*s4_S_Function);

 /* (no update to perform in root model) */
}
```

【図21】

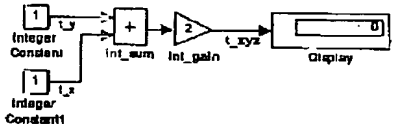
C-code生成
(ラベリング機能)




```
void untitled(void)
{
 s16 t_x;
 s16 t_y;
 s16 t_z;
 s16 t_xyz;

 /* int_sum : s4_S_Function */
 t_z = t_x+t_y;

 /* int_gain : s3_S_Function */
 t_xyz = (s16)(2*t_z);

 /* (no update to perform in root model) */
}
```

【図22】

C-code生成 (グルーピング)



グルーピング無し

```
void untitled(void)
{
 s16 t_x;
 s16 t_y;
 s16 s4_S_Function;
 s16 t_xyz;

 s4_S_Function = t_x+t_y;
 t_xyz = (s16)(2*s4_S_Function);
}
```

グルーピング

```
void untitled(void)
{
 s16 t_x;
 s16 t_y;
 s16 t_xyz;

 /* int_gain : s3_S_Function */
 t_xyz = (s16)(2*t_x+t_y);
}
```

【図23】

グルーピング

(a)

ID	Signal Label
s1_S_Function	t_x

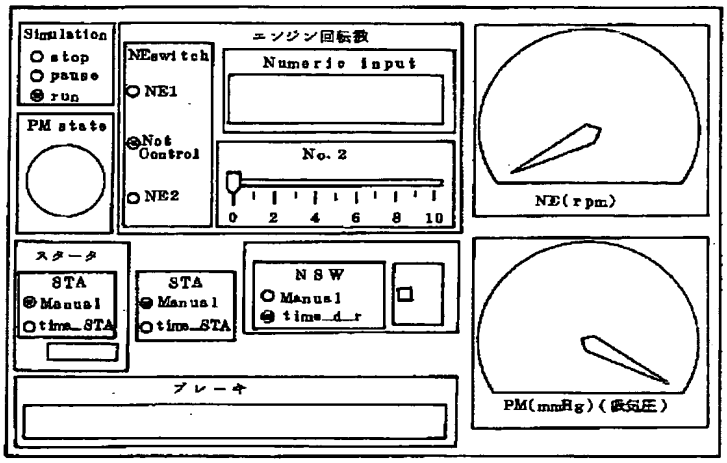
変数名 データベース

(b)

ID	式
s1_S_Function	x1*x2

式のデータベース

【図26】



【図24】